6장. 흐름 제어 구문과 연산자

6.1. 조건문

6.1.1. if 표현식

- 코틀린에서 if는 표현식(expression)

```
fun main(args: Array<String>) {

    val a = 5
    if (a < 10) println("$a < 10")
    //if - else
    if (a > 0  && a <= 10) {
        println("0 < $a <= 10")
    } else if(a > 10 && a <= 20){
        println("10 < $a <=20")
    }else {
        println("$a > 20")
    }
}
```

```
val result=if (a > 10) "hello" else "world"
```

6.1. 조건문

If 표현식

- else 문이 꼭 정의되어야 한다.
- 여러 라인이 작성되는 경우 if 표현식에 의한 데이터는 맨 마지막 라인

```
if(a>10) "hello"//ok
val result2=if(a>10) "hello"//error
```

```
val result2 = if (a < 10) {
    print("hello....")
    10+20
} else {
    print("world...")
    20+20
}
```

```
val result3 = if (a > 10) 20
else if(a > 20) 30
else 10
```

6.1. 조건문

6.1.2. when 표현식

- C 혹은 자바의 switch 구문과 비슷
- when 은 코틀린에서 표현식

```
fun main(args: Array<String>) {
    val a2=1
    when (a2) {
        1 -> println("a2 == 1")
        2 -> println("a2 == 2")
        else -> {
            println("a2 is neither 1 nor 2")
        }
    }
}
```

```
val data1="hello"
when(data1){
    "hello"->println("data1 is hello")
    "world"->println("data1 is world")
    else -> println("data1 is not hello or world")
}
```

6.1. 조건문

- 여러 값을 조건을 표현

```
when(data2){
    10, 20 -> println("data2 is 10 or 20")
    30, 40 -> println("data2 is 30 or 40")
    some() -> println("data2 is 50")
    30 + 30 -> println("data2 is 60")
}
```

- 특정 범위를 조건으로 명시

```
val data3=15
when(data3){
    !in 1..100 -> println("invalid data")
    in 1..10 -> println("1 <= data3 <= 10")
    in 11..20 -> println("11 <= data3 <== 20")
    else -> println("data3 > 20")
}
```

```
val list= listOf<String>("hello","world","kkang")
val array= arrayOf<String>("one","two","three")
val data4="kkang"
when(data4){
    in list -> println("data4 in list")
    in array -> println("data4 in array")
}
```

6.1. 조건문

- 다양한 타입의 데이터에 대한 조건

```
fun testWhen(data: Any){
    when(data){
        1 -> println("data value is 1")
        "hello" -> println("data value is  hello")
        is Boolean -> println("data type is Boolean")
    }
}
```

- if-else 의 대체용

```
val data5=15
when {
    data5<=10 -> println("data5 < 10")
    data5>10 && data5<=20 -> println("10 < data5 <= 20")
    else -> println("data5 > 20")
}
```

6.1. 조건문

- 표현식

```
val data6=3
val result2= when(data6){
    1 -> "1...."
    2 -> "2...."
    else -> {
        println("else....")
        "hello"
    }
}
```

6.2. 반복문

6.2.1. for 반복문

```
fun main(args: Array<String>) {
    var sum: Int=0
    for(i in 1..10) {
        sum += i
    }
    println(sum)
}
```

```
val list = listOf("Hello", "World", "!")
val sb=StringBuffer()
for(str in list) {
    sb.append(str)
}
```

- index 값을 획득하고자 한다면 indices를 이용

```
val list = listOf("Hello", "World", "!")
for (i in list.indices) {
    println(list[i])
}
```

6.2. 반복문

- withIndex()을 이용하여 index와 value를 획득

```
val list = listOf("Hello", "World", "!")
for ((index, value) in list.withIndex()) {
    println("the element at $index is $value")
}
```

for문의 조건

- for (i in 1..100) { //... } // 100까지 포함
- for (i in 1 until 100) { //... } // 100은 포함되지 않음
- for (x in 2..10 step 2) { //... } //2씩 증가
- for (x in 10 downTo 1) { //... }//숫자 감소

```
for(i in 1 until 11 step 2){
    println(i)
}
```

6.2. 반복문

6.2.2. while 반복문

```
fun main(args: Array<String>) {
    var x=0
    var sum1=0
    while (x < 10) {
        sum1 += ++x
    }
    println(sum1)
}
```

6.2. 반복문

6.2.3. break와 continue문, 그리고 라벨

```
fun main(args: Array<String>) {
    var x2=0
    var sum2=0
    while(true){
        sum2 += ++x2
        if(x2==10) break
    }
    println(sum2)

}
```

6.2. 반복문

```
for (i in 1..3) {
    for (j in 1..3) {
        if (j>1) break
        println("i : $i , j : $j")
    }
}
```

```
aaa@ for (i in 1..3) {
    for (j in 1..3) {
        if (j>1) break@aaa
        println("i : $i , j : $j")
    }
}
```

6.3. 연산자

6.3.1. 대입 연산자

연산자	사용법	설명
=	A=B	B 값을 A에 대입

6.3.2. 산술 연산자

연산자	사용법	설명
+	A+B	A와 B의 값을 더하기
-	A-B	A에서 B를 빼기
*	A*B	A와 B를 곱하기
/	A/B	A에서 B를 나누기
%	A%B	A를 B로 나눈 나머지

6.3.3. 전개 연산자

연산자	사용법	설명
*	*A	A 배열의 데이터를 나열

6.3. 연산자

```
fun main(args: Array<String>) {
    val array1= arrayOf(10, 20, 30)

    val list1= asList(1, 2, array1[0], array1[1], array1[2], 100, 200)
    list1.forEach({println(it)})
}
```

```
val array1= arrayOf(10, 20, 30)

val list2=asList(1, 2, *array1, 100, 200)
list2.forEach({println(it)})
```

- 개발자가 직접 정의한 함수를 이용할때도 사용

```
fun some(vararg a: String){
    val iterator=a.iterator()
    while(iterator.hasNext()){
        println(iterator.next())
    }
}

fun main(args: Array<String>) {
    val array3= arrayOf<String>("hello","world")
    some(*array3)
}
```

6.3. 연산자

- 전개 연산자는 배열에 적용

```
val list3= listOf<String>("a","b")
some(*list3.toTypedArray())
```

6.3.4. 복합 대입 연산자

연산자	사용법	설명
+=	A+=B	A와 B의 값을 더한 값을 A에 할당
-=	A-=B	A에서 B를 뺀 값을 A에 할당
*=	A*=B	A와 B를 곱한 값을 A에 할당
/=	A/=B	A에서 B를 나눈 값을 A에 할당
%=	A%=B	A를 B로 나눈 나머지 값을 A에 할당

6.3.5. 증감 연산자

연산자	사용법	설명
++	A++, ++A	A값에 1을 더해 결과 값을 A에 할당
--	A--, --A	A값에서 1을 빼 결과 값을 A에 할당

6.3. 연산자

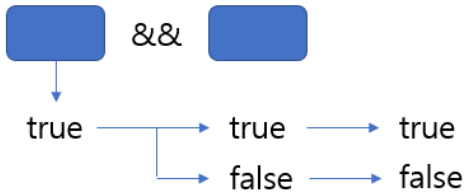
```
fun main(args: Array<String>) {
    var data=10
    var result1 = data++
    println("result=$result1, data=$data")
}
```

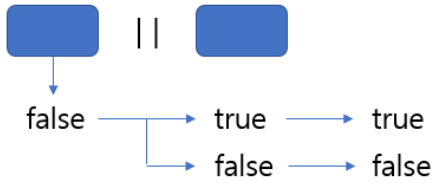
```
var data3=10
var result3 = ++data3
println("result3=$result3, data3=$data3")
```

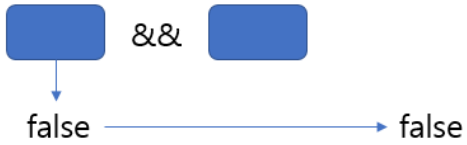
6.3.6. 논리 연산자

연산자	사용법	설명
&&	A && B	A와 B가 모두 true라면 결과값이 true,
||	A || B	A와 B가 하나라도 true이면 결과값은 true
!	!A	A가 true이면 결과값은 false, false이면 결과값은 true

6.3. 연산자







6.3.7. 일치 연산자

연산자	사용법	설명
==	A == B	A와 B의 값을 비교, 같은 값이라면 결과값이 true,
!=	A != B	A와 B의 값을 비교, 값이 다르다면 결과값은 true
===	A === B	A와 B가 같은 객체인지를 비교, 같은 객체면 결과값은 true
!==	A !== B	A아 B가 같은 객체인지를 비교, 다른 객체이면 결과값은 true

- ==을 structural equality, ===은 referential equality
- ==은 값에 대한 비교이고 ===은 객체에 대한 비교
- 일반 객체인지, 기초 데이터 타입의 객체인지에 따라 차이
- ? 에 의해 nullable로 선언되었는지에 따라 차이

6.3. 연산자

- 일반 객체 이용

```
fun main(args: Array<String>) {
    class User
    val user1=User()
    val user2=User()
    val user3=user1
    println("user1==user2 is ${user1==user2}")//false
    println("user1===user2 is ${user1===user2}")//false
    println("user1==user3 is ${user1==user3}")//true
    println("user1===user3 is ${user1===user3}")//true
}
```

- 일반 객체들은 ?에 의해 nullable로 선언

```
val user4=User()
val user5: User?=user4
println("user4==user5 is ${user4==user5}") //true
println("user4===user5 is ${user4===user5}") //true
```

6.3. 연산자

- 기초 데이터 타입

```
val int1=Integer(10)
val int2=Integer(10)
val int3=int1
println("int1==int2 is ${int1==int2}") //true
println("int1===int2 is ${int1===int2}") //false
println("int1==int3 is ${int1==int3}") //true
println("int1===int3 is ${int1===int3}") //true
```

```
val data1: Int=10
val data2: Int=10
println("data1==data2 is ${data1==data2}") //true
println("data1===data2 is ${data1===data2}") //true
```

- ?에 의해 nullable로 선언되면 코틀린에서는 객체를 한번더 boxing

```
val data3=1000
val data4=1000
val data5: Int?=1000
val data6: Int?=1000
println("data3==data4 is ${data3==data4}") //true
println("data3===data4 is ${data3===data4}") //true
println("data5===data6 is ${data5==data6}") //true
println("data5===data6 is ${data5===data6}") //false
```

6.3. 연산자

```
val boxed1:Int? = 127
val boxed2:Int? = 127
val boxed3:Int?=128
val boxed4:Int?=128
println("boxed1===boxed2 is ${boxed1===boxed2}") //true
println("boxed3===boxed4 is ${boxed3===boxed4}") //false
```

```
val double1: Double?=10.0
val double2: Double?=10.0
println("double1==double2 is ${double1==double2}") //true
println("double1===double2 is ${double1===double2}") //false
```

- 기초 데이터 타입이 아닌 일반 클래스의 객체는 ==, ===의 차이가 없다. ?에 의해 boxing이 되든 안되든 동일 reference를 참조하면 true, 다른 객체이면 false
- 기초 데이터 타입의 변수 선언시 자바의 Wrapper 클래스(Integer 같은)를 직접 이용해 생성하면 객체가 생성되는 것임으로 ==은 값을, ===은 객체를 비교
- 기초 데이터 타입의 변수 선언시 Wrapper 클래스 이용없이 Int, Double 등으로 이용하면 ==, === 모두 값 비교
- ? 에 의해 선언된 기초 데이터 타입의 변수는 내부적으로 boxing 되어 객체가 만들어 진다.
- -128~127 까지의 값은 ?에 의해 boxing 되더라도 자바 내부에서 값이 같으면 동일 객체를 리턴하게 된다.

6.3. 연산자

6.3.8. 비교 연산자

연산자	사용법	설명
<	A < B	A가 B 보다 작으면 true
>	A > B	A가 B 보다 크면 true
<=	A <= B	A가 B 보다 작거나 같으면 true
>=	A >= B	A가 B 보다 크거나 같으면 true

6.3.9. 범위 연산자

연산자	사용법	설명
..	A..B	A부터 B까지의 수를 묶어 범위 표현

6.3.10. Null 안전 관련 연산자

연산자	사용법	설명
?	val a: Int?	a 변수를 nullable로 선언
?:	A ?: B	A가 null이면 B 실행
?.	A?.length	A가 null이면 결과 값이 null, null이 아니면 length
!!	A !! B	A가 null 이 아닌경우만 B 실행. null이면 Exception 발생

6.4. 연산자 재정의

6.4.1. 연산자 재정의 방법

- 연산자 재정의는 함수의 재정의를 통해 작성

```
fun main(args: Array<String>) {
    val a: Int = 10

    val b: Int = 5

    val result1: Int = a + b

    val result2: Int = a.plus(b)

    println("result1 : $result1 ... result2 : $result2")
}
```

6.4. 연산자 재정의

```
data class MyClass(val no: Int){
    operator fun plus(arg: Int): Int {
        return no - arg
    }
}

operator fun MyClass.minus(arg: Int): Int{
    return no + arg
}

class Test(val no: Int) {
    operator fun plus(arg: Int): Int {
        return no - arg
    }
}

fun main(args: Array<String>) {
    val obj: MyClass = MyClass(10)

    val result1 = obj + 5
    val result2 = obj - 5

    println("result1 : $result1 .. result2 : $result2") //result1 : 5 .. Result2 : 15

    println("${Test(30) + 5}") //25
}
```

6.4. 연산자 재정의

6.4.2. 연산자 함수

Expression	Translated to
+a	a.unaryPlus()
-a	a.unaryMinus()
!a	a.not()

Expression	Translated to
a++	a.inc()
a--	a.dec()

Expression	Translated to
a + b	a.plus(b)
a – b	a.minus(b)
a * b	a.times(b)
a / b	a.div(b)
a % b	a.rem(b), a.mod(b)(deprecated)
a..b	a.rangeTo(b)

Expression	Translated to
a in b	b.contains(a)
a !in b	!b.contains(a)